The first stable release of Qucs-S and advances in XSPICE model synthesis

Mike Brinson [1], mbrin72043@yahoo.co.uk.
Vadim Kuznetsov [2], ra3xdh@gmail.com

[1] Centre for Communications Technology, London Metropolitan University, UK

[2] Electronic Engineering Department, Bauman Moscow Technical University, Russia

Presented at the Spring MOS-AK Workshop at DATE,
Lausanne, March, 31, 2017

Main features of Qucs-S package

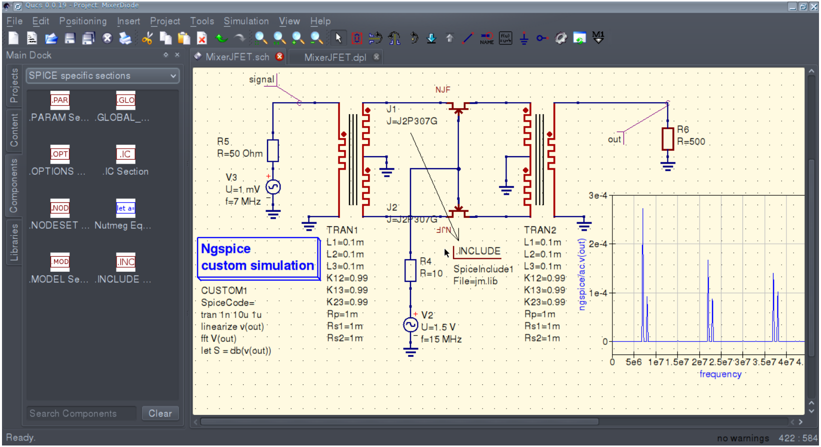
Qucs-S 0.0.19 is the first release of the Qucs-S: unofficial spin-off of Qucs. Main features:

- Ngspice, XYCE, SpiceOpus, and Qucsator user-selectable backends;
- Backward compatible with Qucs by the component types and simulations
- Direct support of SPICE models from components datasheets. SPICE model could be added to schematic without any adaptation.
- Basic SPICE components: RCL, BJT, MOSFET, JFET, MESFET, switches;
- Advanced SPICE components: B-sources and RCLs, transmission lines;
- Direct support of SPICE Modelcards, SPICE sections (.IC, .NODESET); Parametric circuits (.PARAM) and SPICE postrprocessor (Nutmeg)
- Basic (DC, AC, TRAN) and advanced (DISTO, NOISE) SPICE simulations; Single-tone and Multitone Harmonic balanace analysis with XYCE backend;
- Script simulations: Nutmeg script and XYCE script;

Qucs-S subproject website: `https://ra3xdh.github.io/`

The look of Qucs-S main window

- JFET mixer simulation with Qucs-S. Nutmeg script is used for spectrum analysis.

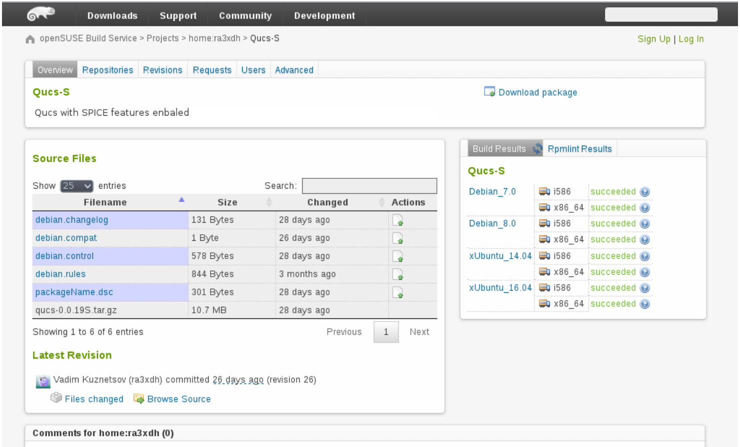


CUSTOM1
SpiceCode=

```
tran 1n 10u 1u
linearize v(out)
fft V(out)
let S = db(v(out))
```

Qucs-S binary packages

- Debian packages are available here: `http://download.opensuse.org/repositories/home:/ra3xdh/`
- Windows Installer: `https://github.com/ra3xdh/qucs/releases/download/0.0.19S/qucs-0.0.19S-setup.zip`



openSUSE Build Service > Projects > home:ra3xdh > Qucs-S

Qucs-S

Qucs with SPICE features enbaled

Source Files

Show 25 entries Search:

Filename	Size	Changed	Actions
debian.changelog	131 Bytes	28 days ago	
debian.compat	1 Byte	26 days ago	
debian.control	578 Bytes	28 days ago	
debian.rules	844 Bytes	3 months ago	
packageName.dsc	301 Bytes	28 days ago	
qucs-0.0.19S.tar.gz	10.7 MB	28 days ago	

Showing 1 to 6 of 6 entries

Latest Revision

Vadim Kuznetsov (ra3xdh) committed 26 days ago (revision 26)

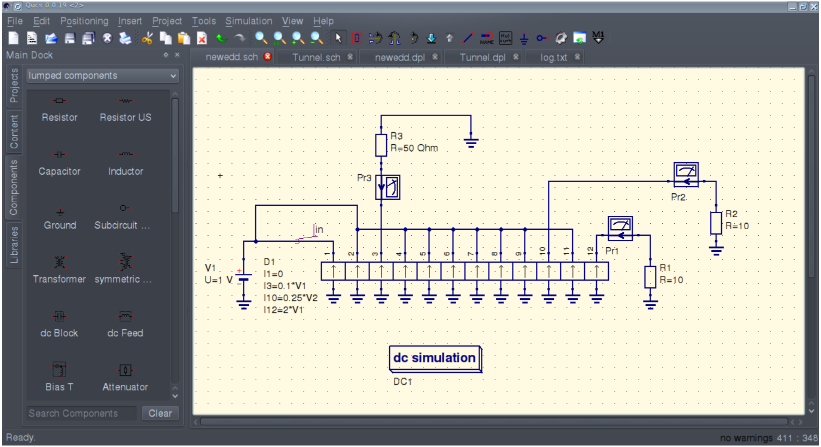
Qucs-S

Debian_7.0	i586	succeeded
	x86_64	succeeded
Debian_8.0	i586	succeeded
	x86_64	succeeded
xUbuntu_14.04	i586	succeeded
	x86_64	succeeded
xUbuntu_16.04	i586	succeeded
	x86_64	succeeded

Comments for home:ra3xdh (0)

New extended EDD

- The original Qucs EDD had only 8 maximum branches allowed. It was extended up to 20 maximum branches to enable construction of more complex compact models.



R3
R=50 Ohm

R2
R=10

R1
R=10

V1
U=1 V

D1
I1=0
I3=0.1*V1
I10=0.25*V2
I12=2*V1

EDD and XSPICE model equations

EDD could be described by the following equations set

- Current equations:

$$I_1 = f_1(V_1, \dots, V_N, I_1, \dots, I_N) \tag{1}$$

$$\dots$$

$$I_N = f_N(V_1, \dots, V_N, I_1, \dots, I_N) \tag{2}$$

- Charge equations:

$$Q_1 = h_1(V_1, \dots, V_N, I_1, \dots, I_N) \tag{3}$$

$$\dots$$

$$Q_N = h_N(V_1, \dots, V_N, I_1, \dots, I_N) \tag{4}$$

XSPICE model equation set

XSPICE model equation set:

- Current equations with capacitance addition:

$$I_1 = f_1(V_1, \ldots, V_N, I_1, \ldots, I_N) + \frac{\partial Q_1(V)}{\partial V_1} \cdot \frac{dV_1}{dt} \tag{5}$$

$$\cdots$$

$$I_N = f_N(V_1, \ldots, V_N, I_1, \ldots, I_N) + \frac{\partial Q_N(V)}{\partial V_N} \cdot \frac{dV_N}{dt} \tag{6}$$

- Partial derivatives of current:

$$\frac{\partial I_1}{\partial V_1} = \frac{\partial}{\partial V_1} \cdot f_1(V_1, \ldots, V_N, I_1, \ldots, I_N) + \frac{\partial Q_1(V)}{\partial V_1} \cdot dt \tag{7}$$

$$\cdots$$

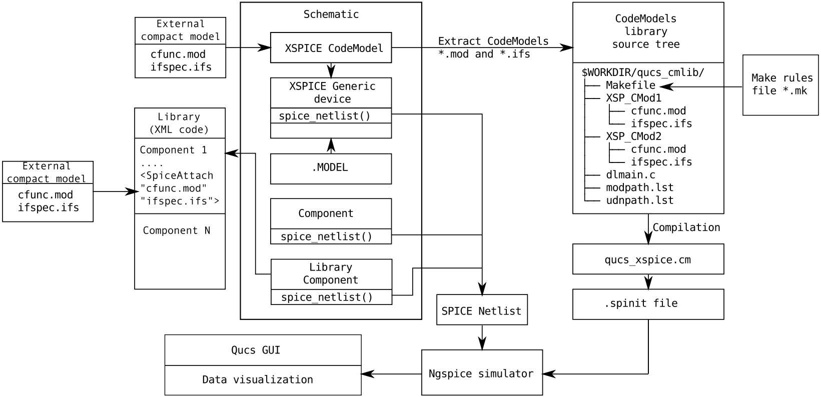
$$\frac{\partial I_N}{\partial V_N} = \frac{\partial}{\partial V_N} \cdot f_N(V_1, \ldots, V_N, I_1, \ldots, I_N) + \frac{\partial Q_N(V)}{\partial V_N} \cdot dt \tag{8}$$

- AC gain matrix

$$(G_{AC}) = \begin{pmatrix} G_{11} & \cdots & G_{N1} \\ \vdots & \ddots & \vdots \\ G_{1N} & \cdots & G_{NN} \end{pmatrix} \tag{9}$$

$$G_{ij} = \frac{\partial I_i}{\partial V_j} + jY_{ij} \tag{10}$$

XSPICE "turn-key" model generation compiler system dataflow diagram



External compact model

cfunc.mod
ifspec.ifs

Schematic

XSPICE CodeModel

XSPICE Generic device

spice_netlist()

.MODEL

Component

spice_netlist()

Library Component

spice_netlist()

Extract CodeModels
*.mod and *.ifs

CodeModels library source tree

```
$WORKDIR/qucs_cmlib/
├── Makefile
├── XSP_CMod1
│   ├── cfunc.mod
│   └── ifspec.ifs
├── XSP_CMod2
│   ├── cfunc.mod
│   └── ifspec.ifs
├── dlmain.c
├── modpath.lst
└── udnpath.lst
```

Make rules file *.mk

Compilation

qucs_xspice.cm

.spinit file

Library (XML code)

Component 1
....
<SpiceAttach "cfunc.mod" "ifspec.ifs">

Component N

External compact model

cfunc.mod
ifspec.ifs

SPICE Netlist

Ngspice simulator

Qucs GUI

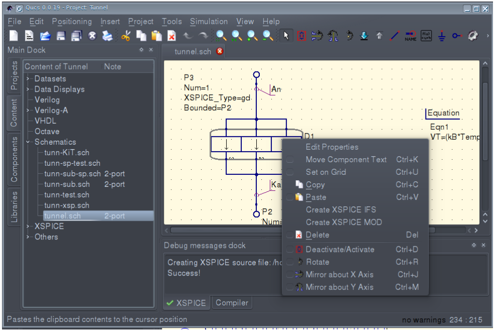
Data visualization

Main features of the XSPICE CodeModel synthesizer

Main features:

- Synthesize XSPICE C-code and interface description from EDD schematic view;
- Access to code synthesizer from right-click on the EDD component;
- Synthesizer generates a pair of MOD and IFS files from a single EDD;
- Automatic recognition of model parameters and dependent variables;
- Automatic symbolic computation of partial derivatives and AC gain matrix using Ginac embedded CAS library;

- XSPICE synthesizer context menu

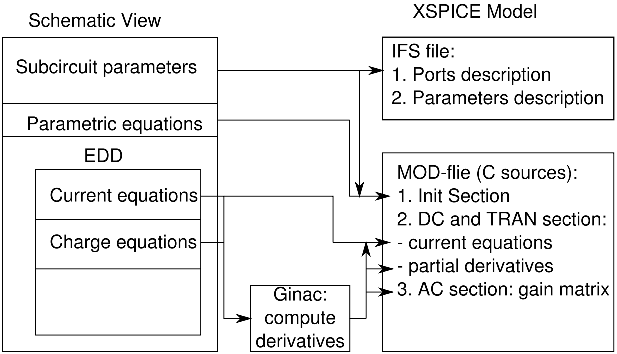


Creating XSPICE source file: /ho
Success!

Pastes the clipboard contents to the cursor position

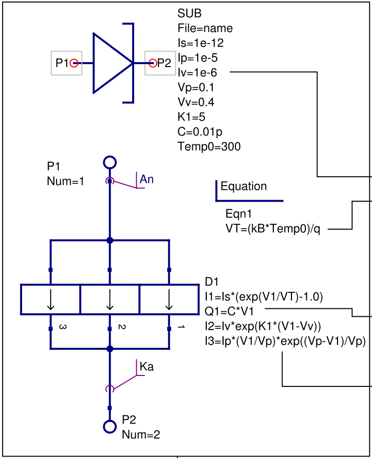
The structure of the XSPICE models synthesizer

- Ginac `http://www.ginac.de/` library is used for symbolic computation of partial derivatives and AC gain matrix;
- Interface description file (*.IFS) is generated from subcircuit symbol or from the EDD and attached equations;
- Model description (C-code *.MOD) is generated from individual EDDs;



Tunnel diode XSPICE model

Schematic view



Press button
"Generate MOD"

XSPICE sources

```
/* XSPICE codemodel tunnel auto-generated template */

#include <math.h>
#include "xspice_mathfunc.h"

void cm_tunnel(ARGS)
{
 Complex_t ac_gain00;
 static double Is,Iv,K1,Vv,Ip,Vp,C,Temp0;
 static double VT;
 static double V1,V1_old;
 double Q0, cQ0; double delta_t;

 if(INIT) {
  Is = PARAM(is); Iv = PARAM(iv);
  K1 = PARAM(k1); Vv = PARAM(vv);
  Ip = PARAM(ip); Vp = PARAM(vp); C = PARAM(c);
  Temp0 = PARAM(temp0);
  VT=8.6173402243760290e-05*Temp0;
 }
 if (ANALYSIS != AC) {
 if (TIME == 0) {
  V1_old = V1 = INPUT(An_Ka);
  Q0=0.0; cQ0=0.0;
 } else {
  V1 = INPUT(An_Ka);
  delta_t=TIME-T(1);
  Q0 = (C)*(V1-V1_old)/(delta_t+1e-20);
  cQ0 = (C)/(delta_t+1e-20);
  V1_old = V1;
 }
  OUTPUT(An_Ka) =  Is*( exp(1.0/VT*V1)-1.0)+
    exp(-( Vv-V1)*K1)*Iv+exp(( Vp-V1)/Vp)*Ip/Vp*V1 + Q0;
  PARTIAL(An_Ka,An_Ka) = Ip*exp(1.0/Vp*(Vp-V1))/Vp+
    1.0/VT*exp(V1/VT)*Is-Ip*exp(1.0/Vp*(Vp-V1))/(Vp*Vp)*V1+
    Iv*K1*exp((V1-Vv)*K1) + cQ0;
 } else {
  ac_gain00.real = Ip*exp(1.0/Vp*(Vp-V1))/Vp+
     1.0/VT*exp(V1/VT)*Is-Ip*exp(1.0/Vp*(Vp-V1))/(Vp*Vp)*V1+
     Iv*K1*exp((V1-Vv)*K1);
  ac_gain00.imag = (C)*RAD_FREQ;
  AC_GAIN(An_Ka,An_Ka) = ac_gain00;
 }
}
```

Two-port testbench model: vacuum triode

- Triode is one of the simplest possible compact models. Triode equations:

$$I_{grid} = 0 \tag{11}$$

$$I_{plate} = \frac{1}{K_g}\left(V_{grid} + \frac{V_{plate}}{\mu}\right)^{1.5} \tag{12}$$

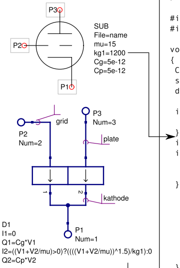
- Model parameters are: μ, K_g, C_{grid}, and C_{plate};
- Additional equations are required to implement XSPICE model (two partial derivatives and AC gain matrix):

$$g_{plate} = \frac{\partial I_{plate}}{\partial V_{plate}} = \frac{1.5}{\mu K_g}\sqrt{\frac{V_{plate}}{\mu} + V_{grid}} \tag{13}$$

$$g_{p.k.} = \frac{\partial I_{plate}}{\partial V_{grid}} = \frac{1.5}{K_g}\sqrt{\frac{V_{plate}}{\mu} + V_{grid}} \tag{14}$$

$$(G_{AC}) = \begin{pmatrix} j\omega C_g & g_{p.k.} \\ 0 & g_{plate} + j\omega C_{plate} \end{pmatrix} \tag{15}$$

Triode EDD implementation and auto-generated XSPICE Code

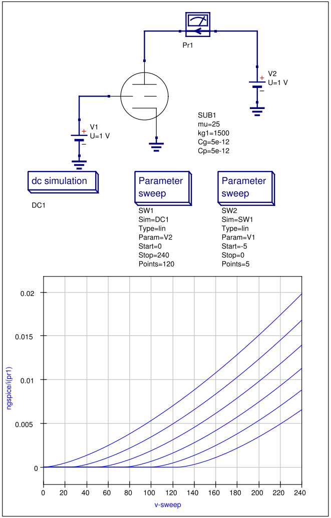


SUB
File=name
mu=15
kg1=1200
Cg=5e-12
Cp=5e-12

D1
I1=0
Q1=Cg*V1
I2=((V1+V2/mu)>0)?((((V1+V2/mu))^1.5)/kg1):0
Q2=Cp*V2

```
/* XSPICE codemodel triode auto-generated template */

#include <math.h>
#include "xspice_mathfunc.h"

void cm_triode(ARGS)
{
 Complex_t ac_gain00, ac_gain01, ac_gain10, ac_gain11;
 static double Cg,mu,kg1,Cp, V1,V2,V1_old,V2_old;
 double Q0, cQ0, Q1, cQ1; double delta_t;

 if(INIT) {
  Cg = PARAM(cg);  mu = PARAM(mu); kg1 = PARAM(kg1); Cp = PARAM(cp);
 }
 if (ANALYSIS != AC) {
 if (TIME == 0) {
  V1_old = V1 = INPUT(grid_kathode); V2_old = V2 = INPUT(plate_kathode);
  Q0=0.0;  cQ0=0.0;  Q1=0.0;  cQ1=0.0;
 } else {
  V1 = INPUT(grid_kathode); V2 = INPUT(plate_kathode);
  delta_t=TIME-T(1);
  Q0 = (Cg)*(V1-V1_old)/(delta_t+1e-20);
  cQ0 = (Cg)/(delta_t+1e-20);
  Q1 = (Cp)*(V2-V2_old)/(delta_t+1e-20);
  cQ1 = (Cp)/(delta_t+1e-20);
  V1_old = V1;  V2_old = V2;
 }
  OUTPUT(grid_kathode) = 0.0 + Q0;
  OUTPUT(plate_kathode) = ((V1+V2/mu)>0)?pow( V1+V2/mu,1.50)/kg1:0.0 + Q1;
  PARTIAL(grid_kathode,grid_kathode) = 0.0 + cQ0;
  PARTIAL(grid_kathode,plate_kathode) = 0.0;
  PARTIAL(plate_kathode,grid_kathode) =
    ((V1+V2/mu)>0)?1.50*Xpow(V1+V2/mu,0.50)/kg1:0.0;
  PARTIAL(plate_kathode,plate_kathode) =
    ((V1+V2/mu)>0)?1.50*Xpow(V2/mu+V1,0.50)/mu/kg1:0.0 + cQ1;
 } else {
  ac_gain00.real = 0.0;
  ac_gain00.imag = (Cg)*RAD_FREQ;
  AC_GAIN(grid_kathode,grid_kathode) = ac_gain00;
  ac_gain01.real = 0.0; ac_gain01.imag = 0.0;
  AC_GAIN(grid_kathode,plate_kathode) = ac_gain01;
  ac_gain10.real = ((V1+V2/mu)>0)?1.50*Xpow(V1+V2/mu,0.5)/kg1:0.0;
  ac_gain10.imag = 0.0;
  AC_GAIN(plate_kathode,grid_kathode) = ac_gain10;
  ac_gain11.real = ((V1+V2/mu)>0)?1.50*Xpow(V2/mu+V1,0.5)/mu/kg1:0.0;
  ac_gain11.imag = (Cp)*RAD_FREQ;
  AC_GAIN(plate_kathode,plate_kathode) = ac_gain11;
 }
}
--
```

DC sweep simulation result



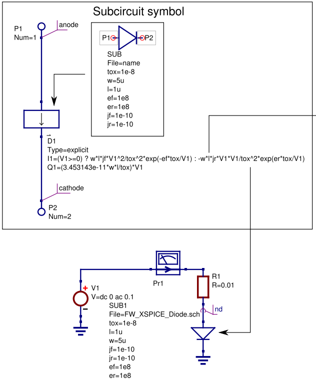
SUB1
mu=25
kg1=1500
Cg=5e-12
Cp=5e-12

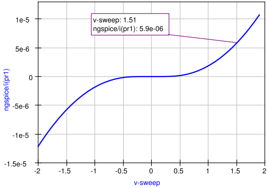
SW1
Sim=DC1
Type=lin
Param=V2
Start=0
Stop=240
Points=120

SW2
Sim=SW1
Type=lin
Param=V1
Start=-5
Stop=0
Points=5

Fowler-Nordheim diode model

Fowler-Nordheim diode



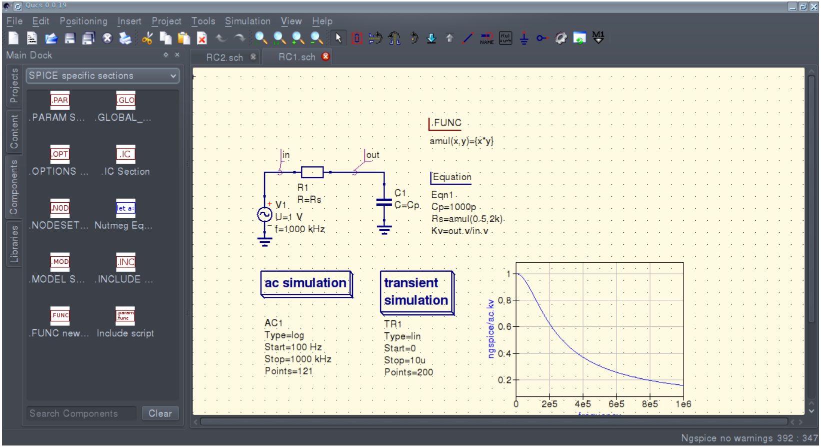


Auto-synthesized XSPICE code

```
/* XSPICE codemodel fw_diode auto-generated template */

#include <math.h>

#define D_0_step(x) (0)
#define step(x) ((x)>0.0?1.0:(((x)==0)?0.5:0.0))

#define Xpow(x,p) pow(fabs((x)),(p))

void cm_fw_diode(ARGS)
{
 Complex_t ac_gain00;
 static double w,l,jf,tox,ef,jr,er;
 static double V1,V1_old;
 double Q0, cQ0;
 double delta_t;

 if(INIT) {
  w = PARAM(w);
  l = PARAM(l);
  jf = PARAM(jf);
  tox = PARAM(tox);
  ef = PARAM(ef);
  jr = PARAM(jr);
  er = PARAM(er);
 }
 if (ANALYSIS != AC) {
 if (TIME == 0) {
  V1_old = V1 = INPUT(anode_cathode);
  Q0=0.0;
  cQ0=0.0;
 } else {
  V1 = INPUT(anode_cathode);
  delta_t=TIME-T(1);
  Q0 = (3.4531430000000001e-11*l/tox*w)*(V1-V1_old)/(delta_t+1e-20);
  cQ0 = (3.4531430000000001e-11*l/tox*w)/(delta_t+1e-20);
  V1_old = V1;
 }
  OUTPUT(anode_cathode) = (V1>=0)?
     jf*(V1*V1)*w*exp(-ef/V1*tox)/(tox*tox)*l:
     -jr*exp(er/V1*tox)*(V1*V1)*w/(tox*tox)*l + Q0;
  PARTIAL(anode_cathode,anode_cathode) = (V1>=0)?
    2.0*1.0/(tox*tox)*l*exp(-tox*ef/V1)*jf*V1*w+1.0/tox*l*exp(-tox*ef/V1)*jf*ef*w:
    -2.0*1.0/(tox*tox)*l*jr*V1*w*exp(tox*er/V1)+1.0/tox*l*jr*er*w*exp(tox*er/V1) + cQ0;
 } else {
  ac_gain00.real = (V1>=0)?
    2.0*1.0/(tox*tox)*l*exp(-tox*ef/V1)*jf*V1*w+1.0/tox*l*exp(-tox*ef/V1)*jf*ef*w:
    -2.0*1.0/(tox*tox)*l*jr*V1*w*exp(tox*er/V1)+1.0/tox*l*jr*er*w*exp(tox*er/V1);
  ac_gain00.imag = (3.4531430000000001e-11*l/tox*w)*RAD_FREQ;
  AC_GAIN(anode_cathode,anode_cathode) = ac_gain00;
 }
}
```

.FUNC entry: user-defined SPICE functions

- .FUNC pseudo-component is placed at the "SPICE specific section group"
- .FUNC entries prepend components description in the auto-generated netlist



.FUNC
amul(x,y)={x*y}

Equation
Eqn1
Cp=1000p
Rs=amul(0.5,2k)
Kv=out.v/in.v

R1
R=Rs

V1
U=1 V
f=1.000 kHz

C1
C=Cp

AC1
Type=log
Start=100 Hz
Stop=1000 kHz
Points=121

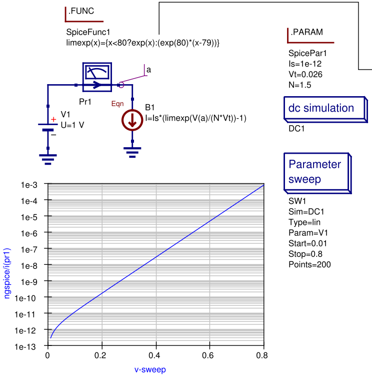
TR1
Type=lin
Start=0
Stop=10u
Points=200

.FUNC entry: user-defined SPICE functions

- Diode model implementation with Ngspice and limexp() function:

$$I = Is\left(\exp\left(\frac{V}{NV_t}\right) - 1\right) \tag{16}$$

- Schematic and auto-generated SPICE netlist:

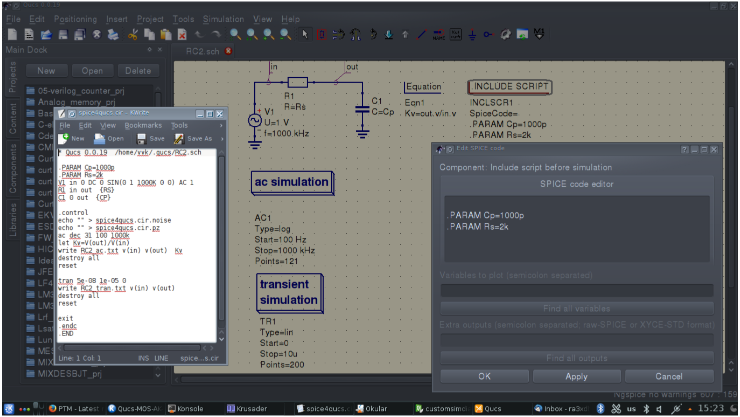


SPICE netlist

```
* Qucs 0.0.19  /tmp/func_demo.sch
.FUNC limexp(x) = {x<80?exp(x):(exp(80)*(x-79))}
.PARAM Is = 1e-12
.PARAM Vt = 0.026
.PARAM N = 1.5
V1 _net0 0 DC 1

VPr1 _net0 a DC 0
B1 a  0  I = Is*(limexp(V(a)/(N*Vt))-1)
.control
echo "" > spice4qucs.cir.noise
echo "" > spice4qucs.cir.pz
dc v1 0.01 0.8 0.00395
write func_demo_dc.txt VPr1#branch v(a)
destroy all
reset

exit
.endc
.END
```

Include scripts: run some SPICE code before components initialized

- Include scripts allow to place some custom SPICE code (parameters, options, function definitions) before components initialization and edit this code manually.



```
* Qucs 0.0.19  /home/vvk/.qucs/RC2.sch

.PARAM Cp=1000p
.PARAM Rs=2k
V1 in 0 DC 0 SIN(0 1 1000K 0 0) AC 1
R1 in out  {RS}
C1 0 out  {CP}

.control
echo "" > spice4qucs.cir.noise
echo "" > spice4qucs.cir.pz
ac dec 31 100 1000k
let Kv=V(out)/V(in)
write RC2_ac.txt v(in) v(out)  Kv
destroy all
reset

tran 5e-08 1e-05 0
write RC2_tran.txt v(in) v(out)
destroy all
reset

exit
.endc
.END
```

Component: Include script before simulation

SPICE code editor

```
.PARAM Cp=1000p
.PARAM Rs=2k
```

Variables to plot (semicolon separated)

Extra outputs (semicolon separated; raw-SPICE or XYCE-STD format)

Conclusion: Plans for future

Possible new directions in XSPICE synthesizer development:

- Synthesize a more complex XSPICE models: EKV, GaN HEMT, etc.;
- A new generation of components: source based components. The C-source is dynamically synthesized and compiled before the simulation;
- Link symbolic computations libraries to XSPICE kernel: perform symbolic computations at simulation time;
- Extend a library pre-synthesized XSPICE models shipped with Qucs-S (currently having the tunnel diode library).

Conclusion: Plans for future

Plans for the next Qucs-S 0.0.20 release:

- Include an XSPICE code synthesizer;
- The support for .FUNC and Include scripts;
- Improvements in XYCE support: new components and .SENS analysis;
- Ngspice digital library;
- Synchronize code base with mainline Qucs and bugfixing;
- Release date scheduled: Summer 2017;

Source code available at:

- Stable and release candidates:
  `https://github.com/ra3xdh/qucs/tree/qucs-s-stable`
- Development branch:
  `https://github.com/ra3xdh/qucs/tree/spice4qucs_current`